Electronic Communications of the EASST
Volume 22 (2009)

Proceedings of the Third International Workshop on Formal Methods for Interactive Systems (FMIS 2009)

Tightly coupled verification of pervasive systems

Muffy Calder, Phil Gray and Chris Unsworth

16 pages

Guest Editors: Michael Harrison, Mieke Massink
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Tightly coupled verification of pervasive systems

Muffy Calder, Phil Gray and Chris Unsworth

Department of Computing Science
University of Glasgow Glasgow G12 8RZ, UK

Abstract: We consider the problem of verifying context-aware, pervasive, interactive systems when the interaction involves both system configuration and system use. Verification of configurable systems is more tightly coupled to design when the verification process involves reasoning about configurable formal models. The approach is illustrated with a case study: using the model checker SPIN [Hol03] and a SAT solver [ES03] to reason about a configurable model of an activity monitor from the MATCH homecare infrastructure [MG09]. Parts of the models are generated automatically from actual log files.

Keywords: formal models, pervasive systems, model checking

1 Introduction

Effective verification of interactive systems has been a significant challenge for both the verification and user interface communities for at least the last two decades [CH97, CRB07]. More recently, the advent of context-aware, pervasive, interactive systems raises the stakes: can we formulate effective verification techniques and strategies to bring reasoning into the design processes of these volatile systems? In particular, we are concerned with systems where the interaction involves both system configuration and system use.

Pervasive systems are characterised by their ability to sense their physical environment and a use of data so gathered both as part of the core application functionality and as a way of modifying system behaviour to reflect changes in the context of use. Amongst the challenges of designing, building and operating such systems is the volatility that this sensor-based context dependency introduces. The set of sensors may themselves come and go and change their behaviour depending upon environmental conditions. Context changes may be difficult to model and predict. In addition, pervasive applications often operate in situations that require practically unpredictable changes to the application functionality itself. For example, a home care system, providing sensor-based monitoring of the cared person's activities and state, may have to be reconfigured to take into account changes in the person's medical condition and their home situation, and consequent changes to the services and sensors needed. For these reasons, system configuration must be treated as an ongoing process throughout the lifetime of a system. It must be modelled and reasoned about in the same way that one would model and reason about normal user interaction with the system.

At a high level of abstraction these context-aware, interactive systems may be regarded as a number of concurrent processes:

$$Agents||Sensors||Outputs||Monitors||Configuration||System$$

where

- *Agents* are (usually, but not exclusively human) users, there may be several types and (possibly overlapping) instances of user, e.g. patient, carer, social worker, etc.
- *Sensors* and *Outputs* quantify physical world data (e.g. thermometer, pressure pad, web-cam), or are outputs devices (e.g. speaker, television screen),
- *Monitors* are high level abstractions of a physical state (e.g. encapsulating predicates about who is in a room, whether or not is it cold),
- *Configurations* are sets of rules, or actual parameters, determining how the system varies according to user preferences and needs,
- the *System* is the underlying computational infrastructure.

While we might traditionally consider the composition $(Agents||Sensors||Outputs||Monitors)$ as the context, i.e. together they reflect a temporal physical context, the $Configuration$ is also a context, in that it is also temporal and affects system behaviour.

In this paper we consider the modelling and verification process for configurable, interactive, context-aware systems in general, and a case study of an event driven system. We begin with a general purpose model of functional behaviour and for this we propose that a (concurrent) process based specification language, temporal logics and reasoning by model checking is a good paradigm, especially when context changes are non-deterministic. In the case study here we develop a general purpose model in Promela, for checking with the SPIN model checker [Hol03]. We then refine the verification problem and develop a specialised model for checking redundacies, using a SAT solver [ES03]. A distinctive feature of this work is parts of the models are generated automatically from actual log files.

In the next section we outline our overall vision for the modelling and verification process. The remainder of the paper is an exploration of one iteration of that process, for a case study. Section 3 introduces the MATCH case study and in section 4 we give an outline of our Promela model. Properties for verification are given in Section 5, where we give an overview of checking for redundant rules in the Promela model. In Section 6 we give an outline of the SAT model for redundancy checking and results of online verification. In Section 7 we consider the more general problem of overlapping left and/or right hand sides of rules, and when these should be interpreted as undesirable. Discussion follows in Section 8 and an overview of related work follows. Conclusions and future work are in Section 10.

2 Modelling and verification process

Traditionally, modelling is a manual process with the starting point of a system specification, or a set of requirements, or, when the system is operational, observations and data. One notable exception is [HS99], where the Promela model is extracted mechanically from call processing software code: a control-flow skeleton is created using a simple parser, and this skeleton is then populated with all message-passing operations from the original code. Our vision for the modelling and verification process is similar in that we aim to more tightly couple the model

and the system, and indeed the results of the verification. Crucial to the process is the notion of configuration and the extraction from the system of configuration details, often stored as log files.

Our vision is illustrated in Figure 1, where ellipses denote agents and rectangles objects. The key feature of our vision is that modelling is tightly coupled with system development and configuration. This is not a waterfall model: activities are concurrent and moreover, while four agent roles are indicated, they may be overlapping or conflated. Briefly, activities are as follows. The end users configure the system, and when configured, (possibly different) users interact with the system, as system and users require, according to the context. The configuration is not static, but may be changed repeatedly. Log files are a representation of the configuration process and are generated by a live system. The formal model depends upon what kind of analysis is required (e.g. functional behaviour, security, performance, etc.) and it is also configured, according to the log files. The model is analysed; the verification results may inform understandings of the end user, the configurer, the designer, and the modeller, though in different ways. For example, the user develops a better cognitive model, the configurer understands how to improve his/her rules, the designer develops a better interface, and the modeller gains insight in to how to modify the model so that verification is more efficient. Note, this is just an example. There may be multiple models and a single agent may have multiple roles as configurer/modeller/user/designer. Verification may be performed off-line or on-line, each of which has its merits. On-line verification can inform users in real-time. On the other hand, off-line verification allows more general results e.g. for all configurations with a certain property, a system proprety holds. This kind of verification can then be used by the designer to constrain allowable interactions or configurations. Finally, recall that agents may not be human at all, for example, the system might autonomously configure itself, or the modeller may be another software process.

Properties may support, for example,

- end user configurations, e.g. *what will happen if I add this rule?* or *how can I notify/detect x in the event of y?*
- modalities, e.g. *are there multiple speech outputs?* or *are there multiple speech inputs only when there are multiple users?*
- hypotheses about resources, e.g. *what happens if a webcam doesn't work?*

In this paper we report on one iteration of the modelling and verification cycle, starting with log files extracted from actual system trials of a prototype system (deployed in the UK and in France).

3 MATCH System

Activity awareness systems constitute an increasingly popular category of pervasive application [MRM09]. Such systems allow groups of users to share information about their current status or recent activities. They have a variety of purposes, ranging from supporting collaborative work through informal social relationships. We have chosen to investigate our approach to verification using one such activity awareness system, the MATCH Activity Monitor (hereafter, MAM), an

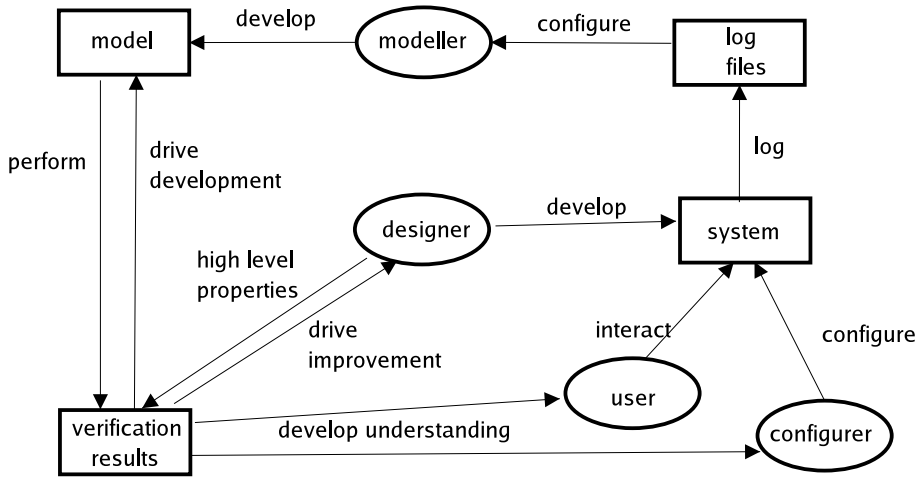

Figure 1: Tightly coupled verification: configurable systems and configurable models

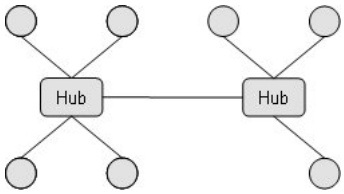

Figure 2: MAM System Architecture

experimental platform designed to explore the challenges of the configuration of activity awareness use to support of home-care [MG09].

A MAM system consists of one or more hubs (UMPC-based subsystems supporting a rich set of inputs and message types) each of which is connected to a set of satellites (web-based clients offering a limited set of inputs and message types) and other hubs, illustrated via the architecture diagram in Figure 2. Typically, a hub will reside in the home of a person requiring care while the satellites are used by carers, clinicians, family and friends.

Each hub, placed in the home of a cared person, can communicate with a set of web-based clients and with other hubs. A MAM hub supports a set of up to eight monitoring tasks, each of which involves the generation of messages based on user-generated or sensor-generated input indicating an event or activity. Monitoring tasks are defined by rules that specify an event or activity to be reported plus the destination and form of the reporting message. For example, a rule might state that use of Tony's coffee cup (captured via an appropriate sensor[1]) should be reported to me (i.e., the hub in my home) via a speech message. Currently, MAM supports a variety of data sources, message destinations and message modalities (e.g., speech, graphics,

[1] MAM uses a JAKE sensor pack for simple movement sensing, while the JAKEs more powerful sibling, the SHAKE, provides richer sensing capabilities and tactile feedback [JAK, SHA].

touch, etc.). A full list is given in Figure 3.

Each MAM hub can support up to eight monitoring tasks, each of which is specified explicitly as a monitoring rule[2]. In addition to simple <input source> <destination, modality> rules, it is also possible to specify combinations of inputs (e.g., a button press or an appointment) or message modalities (e.g., speech and graphics). Rules may also have a guard condition; currently MAM only supports a location condition such that the message is sent if someone is sensed near a specified location.

A user may also choose a system-generated recommendation of the input, destination or modality. The recommendation can be used in an automatic or semi-automatic mode. In the former case, the system will choose the input, destination or modality most commonly associated with the other parameters, based on a history of logged configurations. In the latter case, the system will offer a ranked list of choices, based again on frequency of association, from which the user must select one.

A user interface is supplied for specifying monitoring task rules. If a user is not interacting with the MAM Hub, it operates a digital photo frame application that displays the user's photos in order to make it a non-intrusive part of the user's home. To configure the hub, a user touches the screen and the photo application fades away, replaced by the MAM application, from which the configuration screen is accessible. Figure 4 shows a typical rule configuration. Note the eight tabs to the left, one for each rule; rule 1 is selected. The rule configuration view is divided into a left-hand panel for specifying input and a right-hand panel for destination and modality. In this case the blue and red buttons on my hub (left-hand panel) have been selected to create messages to be sent to Lucy's machine(s) (right-hand panel). The large vertical green button on the right of the panel is the on-off toggle switch for the rule; when green, the rule is active and when red the rule is inactive.

Even with the rather limited set of inputs, destinations and message types, the configuration space (i.e., number of different possible rules) is rather large (1.07E+301). In addition, not all configurations are desirable. It is possible to create redundant rules, which can be a problem given the restriction on the number of rules allowed. Also, some configurations may cause difficulties for the user: two speech messages delivered at the same time will be impossible to understand. These configuration challenges provide a motivation for verification that can be used both to guide a designer (in exploring the design of the configuration options offered to a user) and to help an end user (in creating a set of rules that both meet their needs and are understandable and maintainable).

4 General model

From a modelling perspective, the MAM system is an event driven rule-based pervasive system. Events include (but are not restricted to) direct user interaction with the hub, such as pushing buttons, and indirect user interaction such as movement captured by a webcam or external actions such as messages received from other users. Rules dictate how the system will react to events. We note that from a modelling perspective, there is no distinction between a user interaction and

[2] This limitation, amongst others, is intentional and based on empirical evidence, to limit the complexity of the application.

Input Sources	
Calendar	An online calendar scheduling system reports upcoming appointments
Accelerometer	Small custom-built Bluetooth accelerometers can be placed around the home (e.g., on a phone, teacup, or door) or on a person, in order to detect movement-signalled activity of the instrumented thing/person. This is performed using JAKE and SHAKE devices [WMH07].
Webcam movement events	Fixed and wireless webcams can be used to provide motion detection. This allows for room occupancy to be detected and reported.
User-generated text	Users can key in their current activity, mood or needs explicitly using an on-screen keyboard.
Abstract Buttons	A user may select an abstract button to which no particular meaning has been assigned in advance by the developers of the system (i.e. the red square). The user may negotiate with other people to assign a particular meaning to these buttons. This concept is derived from MarkerClock [RM07] that uses a similar abstract marker feature.
Message Destinations	
Local Hub	Messages are directed to one of the output devices associated with the local machine.
Registered Users	Messages are directed to specified users; the message will be sent to their hub, if they have one, or to their registered web-based client(s).
Modalities	
Graphical	Notice of an activity is briefly overlaid on top of the hub photoframe; an icon indicates that there is an unread message waiting. Additionally, the message will be added to a scrollable list of messages that is permanently available.
Speech	The content of the message is rendered into VoiceXML and played through any of the devices speakers.
Non-speech audio	A selection of auditory alerts is provided, such as nature sounds as well as more familiar alert noises. Each set of sounds contains multiple .wav files, each of which is mapped to a particular type of alert. As with speech, this can be directed to any distinct speaker.
Tactile	The SHAKE device (but not the JAKE) is equipped with an inbuilt vibrotactile actuator that can be activated. Vibration profiles (i.e. vibrate fast-slow-fast. slow-fast-slow) can be used to distinguish between different types of activity.
Email	Activity messages can be delivered to one or more email addresses that the user can specify.

Figure 3: MAM Activity Monitoring Task Parameters

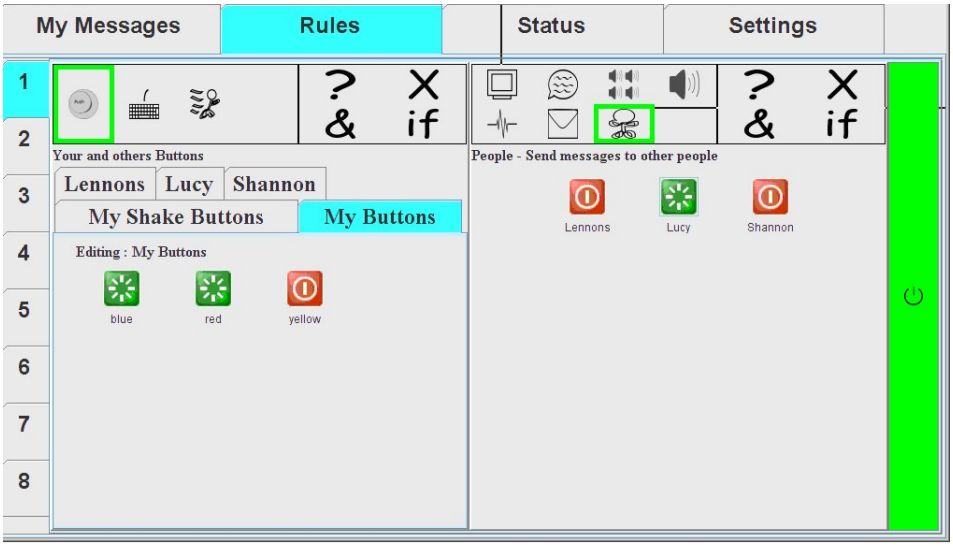

Figure 4: Sample task configuration screen

change of context. While there is intent associated with the former, from a modelling point of view both are simply aspects of state that may be captured by propositions (whose validity may be temporal).

Promela [Hol03] is a high-level, state-based, language for modelling communicating, concurrent processes. It is an imperative, C-like language with additional constructs for non-determinism, asynchronous and rendezvous (synchronizing) communication, dynamic process creation, and mobile connections, i.e. communication channels can be passed along other communication channels. The language is very expressive, and has intuitive Kripke structure semantics.

Our model is centred around a single hub that can take input from one or more satellites or additional hubs. As a result we have a single rule set, which in this case is static. However, it would be a simple matter to extend the model to include multiple hubs and rule sets and dynamic rule sets.

As we are considering a configurable system, the model is designed to reflect this. The system behaviour is separated from the system configuration. System behaviour refers to the actions of the available input and output devices. System configuration refers to the current active rule set.

4.1 System

Each input device is represented by a global variable and a process. For example, a button press is represented by a single bit variable and a process that can arbitrarily assign the values 0 or 1.

Movement sensors such as a jake, shake or webcam, are represented as an integer variable. The movement process will arbitrarily assign a values 0-3, where 0 represents no movement and 1,2 and 3 represent low, medium and high levels of movement respectively. Text based inputs, such as messages from other hubs, are represented as an mtype variable. The associated process will arbitrarily assign values representing one of the users in the system or a null value. These

processes act as sources of events for the system.

Output devices act as sinks within the system. In the model they are represented as global variables or channels, upon which messages are placed. The associated process for an output device is called when a value is assigned to the variable/posted in the channel. The process then resets the variable or reads the message off the channel.

In future versions of the model it may be useful to include users and/or multiple hubs and rule sets in the system. In this case, the input variables will be directly modified by output processes from other hubs and/or as a direct result of a user action.

4.2 Rules

Rules are taken directly from a MAM system via the log. An excerpt from a log file can be seen in Figure 5, included as an illustration of the content of a log file (and not to be read in detail!).

```
uk.org.match_proj.osgi.EvalFunc.ManualEachComponentInputSelectionEvaluationFunction(selected=["Personal Messages"]){}
uk.org.match_proj.osgi.EvalFunc.UnionOutputApprovalEvaluationFunction()
{uk.org.match_proj.osgi.EvalFunc.ManualEachComponentOutputSelectionEvaluationFunction(selected=["GUI","Twitter"])
{}uk.org.match_proj.osgi.EvalFunc.ManualEachPersonSelectionEvaluationFunction(selected=["Doms","Lionel","Anne"]){}}
uk.org.match_proj.osgi.EvalFunc.ManualGroupedPersonSelectionEvaluationFunction(selected=["Everyone"]){}
uk.org.match_proj.osgi.EvalFunc.ManualEachComponentOutputSelectionEvaluationFunction(selected=["GUI"]){}
uk.org.match_proj.osgi.EvalFunc.ManualEachPersonSelectionEvaluationFunction(selected=["Caroline","Lionel","Anne"]){}
uk.org.match_proj.osgi.EvalFunc.ManualEachComponentOutputSelectionEvaluationFunction(selected=["Speech"]){}
uk.org.match_proj.osgi.EvalFunc.ManualEachComponentInputSelectionEvaluationFunction(selected=["Jake Movement"]){}
uk.org.match_proj.osgi.EvalFunc.UnionOutputApprovalEvaluationFunction()
{uk.org.match_proj.osgi.EvalFunc.ManualEachPersonSelectionEvaluationFunction(selected=["Caroline","Lionel","Anne"])
{}uk.org.match_proj.osgi.EvalFunc.ManualEachComponentOutputSelectionEvaluationFunction(selected=["GUI","Twitter"]){}}
uk.org.match_proj.osgi.EvalFunc.ManualEachComponentInputSelectionEvaluationFunction(selected=["Work Messages"]){}
uk.org.match_proj.osgi.EvalFunc.ManualEachPersonSelectionEvaluationFunction(selected=["Caroline","Anne"]){}
```

Figure 5: Excerpt from a MAM log file.

In the MAM system, rules are defined as evaluation functions that return input or output devices. Each rule consists of an input and an output evaluation function. The combination of function name and the list of parameters determine how they are to act. Both input and output functions can be composed. The input composition operator acts as a disjunction, meaning that an event will be triggered if either evaluation function is true. However, the output composition function acts as a conjunction, meaning that if the union output function is triggered then the result of both evaluation functions will be used.

The evaluation function rule set is then expressed as an informal natural language rule set. An example is shown in Figure 6. While this step is not strictly necessary, it can be helpful to have the rules expressed in a more readable format.

The rules are then expressed as Promela statements. Each rule is expressed as a conditional statement, consisting of a guard and a compound statement. Therefore, in Promela we represent a rule as a single statement $C \rightarrow A$, where C is a guard statement made up of a disjunction of statements representing the condition of the rule and A is compound statement consisting of a sequence of statements representing the action. For example, the rule "when the red console button is pressed play the doorbell earcon[3] on the hub speaker" maps to the Promela statement "$(this.red > 0) \rightarrow this.speaker!earcon\_doorbell$". Rules can also be context sensitive. For example, "If the red button is pressed then, if the webcam has recently detected movement inform

[3] An earcon is a short meaningful audio segment.

If the red or blue buttons are pressed	then	play the rocket earcon
If my webcam detects movement	then	display a pop-up message on my screen and display a message on the screen list
If I receive a message from Bill or Bill presses his red button	then	inform me using synthesised speech
If I receive a message from Bill	then	send a vibration message via the shake
If the red button is pressed	then	send a message to Bill and inform me using synthesised speech
If the shake senses movement	then	send a vibration message via the shake
If Bill presses his red button	then	inform me using synthesised speech
If the yellow button is pressed	then	send a vibration message via the shake

Figure 6: Example rule set.

me with synthesised speech else send me an e-mail". In this case the definition of "recently" is a system parameter.

An example of a Promela representation of a rule set can be seen in Figure 7. In this example there are 2 hubs, one that belongs to the user and one that belongs to Bill. In the rule set the user's hub is referred to as *this* and Bill's hub is *Billh*. This is because in the model a hub is represented by a bespoke variable type.

```
proctype rules()
{
do
  :: (this.red == 1) || (this.blue == 1) -> this.audio_out!ec_rocket;
  :: (this.webcam > 2) -> this.screen_popup = me; this.screen_list = me;
  :: (this.text_in == Bill || Billh.red == 1) -> this.audio_out!speech;
  :: (this.text_in == Bill) -> this.shake_out = 1;
  :: (this.red > 0) ->  Billh.text_in = me; this.audio_out!speech;
  :: (this.shake_in_m > 1) -> this.shake_out = 1;
  :: (Billh.red > 0) -> this.audio_out = speech;
  :: (this.yellow > 0) -> this.shake_out = 1;
od
}
```

Figure 7: Promela representation of example rule set.

5 Properties

We now explore a number of issues in the MAM system that may benefit from formal verification.

5.1 Redundant rule detection

As the rules may be added by non-expert users, some may have overlapping or repeated definitions. It would be advantageous if the system could detect such redundant rules to be able

to streamline the system. This could provide feedback to the user and/or allow the system to remove redundant rules from the active rule set.

5.2 Modalities

The input and output devices can be classified by their modalities. For example, earcons and speech are sound, screen pop-ups and text messages are visual and vibration alerts are tactile. The acceptability of a system for a user may depend on the correct use of the different modalities. For example, multiple simultaneous audio outputs may confuse a user and result in the loss of messages. Visual output devices should be avoided for severely visually impaired users, however, it may still be appropriate to use them to notify carers. Overuse of tactile devices may result in the user being unable to differentiate between different types of messages.

5.3 Priorities

Currently MAM does not hold information on the relative priorities of rules/messages. However, a user is likely to be more concerned with certain messages than others. It would therefore be useful to check that the output from a given rule has a greater chance of being received by the appropriate target. For example, high priority messages should be distinct from lower priority messages. If a high priority message uses an earcon, then no other message should use a similar sounding earcon.

5.4 Verification results

We now expand on the redundant rule detection problem using the model checker SPIN, which verifies (or not) properties expressed in the logic LTL (linear temporal logic).

An LTL property can be derived easily from the Promela representation of a rule. In the general form, for a rule r of form $C \rightarrow A$, where C is a guard and A is a sequence of statements, the associated property is informally described by: the action will always eventually occur after the condition becomes true (recall, conditions are disjunctions and actions are conjunctions. More formally, we define the mapping as $f(r) = \Box(f(C) \rightarrow \Diamond f(A))$, where $f()$ maps guards and assignments to propositions in the obvious way.

For example, the rule $(this.yellow > 0) -> this.shake\_out = 1$ would map to the LTL property $\Box((this.yellow > 0) \rightarrow \Diamond(this.shake\_out = 1))$.

We define $f(r)$ as the property that the action associated with rule r will always eventually occur after the condition of r becomes true. A rule r in the rule set R is redundant if for model $\mathscr{M}(R|r)$, which represents R without r, $f(r) \models \mathscr{M}(R|r)$. A rule set R contains no redundant rules if $\forall r f(r) \not\models \mathscr{M}(R|r)$.

Redundancy checking was implemented with a realistic rule set (shown in Figure 8) taken from an actual log file from the MAM system. Each rule was tested in-turn for redundancy. Rule r7 was found to be redundant. Verification times varied from around 12 minutes to 34 minutes. The search depth was between 4 and 6 million, and the number of states explored was between 65 and 100 million. Clearly if this is to be used for real-time verification then significant improvements need to be made in the model efficiency and/or the verification techniques employed. However,

these results do serve as a proof of concept.

r1	$((this.red == 1)\|\|(this.blue == 1)$ $\|\|(this.yellow == 1))$	$\rightarrow$	$this.audio\_out = ec\_rocket$
r2	$(this.webcam > 2)$	$\rightarrow$	$this.screen\_popup = me$; $this.screen\_list = me$
r3	$(this.text\_in == Bill)\|\|(Billh.red == 1)$ $\|\|(Billh.blue == 1)\|\|(Billh.yellow == 1)$	$\rightarrow$	$this.audio\_out = speech$; $this.screen\_list = me$
r4	$(this.text\_in == Bill)$	$\rightarrow$	$this.shake\_out = 1$
r5	$(this.red > 0)$	$\rightarrow$	$Billh.text\_in = me$
r6	$(this.shake\_in\_m > 1)$	$\rightarrow$	$this.shake\_out = 1$
r7	$(Billh.red > 0)$	$\rightarrow$	$this.audio\_out = speech$
r8	$(this.yellow > 0)$	$\rightarrow$	$this.shake\_out = 1$

Figure 8: Rule set used in experiments.

6 Specialised model

From Section 5.4 it can be seen that our current general Promela model of the MAM system is not sufficiently efficient to detect redundancy fast enough to provide feedback to a system configurer in real-time. In this section we show how redundancy can be modelled and solved efficiently with a specialised SAT solver [ES03]. SAT solvers check satisfiability of propositional formulae (usually written in disjunctive normal form). Though in general NP-complete, SAT solvers are highly efficient for many practical applications. The SAT model developed here uses literals to represent input devices, output devices and the rules.

6.1 Literals - inputs and outputs

Each simple input type is represented by a single literal. For example, a literal represents a button press or receiving a message from someone. Similarly, simple output types are also represented as literals. More complex input functions such as movement, which has low, medium and high inputs, can be represented as one literal per input value. A clause then needs to be added to ensure the input values are consistent. For example, if the literal for a movement level high is true, then both medium and low should also be true. To ensure this, the clauses $low \vee \neg medium$ and $medium \vee \neg high$ are added, which will need to be done for each movement detection device. However, if only one rule takes input from a movement detection device, then the input can be treated as a simple input device and the clause can be omitted. Classes of output, such as the MAM auditory icon class "nature", which consists of the auditory icons {wave, forest, wind}, can be modelled in one of two ways. If none of the individual auditory icons from this group are used as output for other rules, then the group can be represented as a single literal. Otherwise, each of the class members will be represented as a single literal and there is an additional literal for the class. For example, the class "nature" will be represented by $wave \vee forest \vee wind \vee \neg nature$

1.	$\forall r \in R$	$\forall c \in r$	$r_i \vee \neg c$
2.	$\forall r \in R$	1	$c_1 \vee c_2 \vee \cdots \vee c_n \vee \neg r_i$
3.	$\forall r \in R$	$\forall a \in r$	$\neg r_i \vee a$
4.	$\forall a \in R$	1	$r_1 \vee r_2 \vee \cdots \vee r_n \vee \neg a_i$

Figure 9: Clauses required to represent a given rule set R

6.2 Clauses - rules

Each rule is represented by a literal r_i and a set of clauses as described in Figure 9. There are four types of clauses associated with a rule, as follows. The literal r_i being assigned the value true indicates that rule i has been triggered. A clause $r_i \vee \neg c$ is added for each condition c that triggers rule i. To ensure r_i is not true if none of its conditions are met, the following clause is added $c_1 \vee c_2 \vee \cdots \vee c_n \vee \neg r_i$. If rule i is triggered then the appropriate outputs must be set to true, thus the clause $\neg r_i \vee a$ is added for each action a associated with rule i. Finally, to ensure that actions are only taken if triggered by a rule, the clause $r_1 \vee r_2 \vee \cdots \vee r_n \vee \neg a_i$ is added, where r_1 to r_n are all the rules that can trigger action a_i.

6.3 Rule redundancy

To use the above model to detect redundant rules, we need to solve the model once for each atomic condition in the rule, to check if atomic condition c from rule r_i is redundant. We add a clause for each input literal, setting the literal related to c to true and all the rest to false. All literals that represent the actions from rule r_i are set to true. All clauses related to the rule being checked are removed. If the resultant model is satisfiable then condition c from rule r_i is redundant, if all conditions associated with rule r_i are redundant, then r_i is redundant.

6.4 Implementation and complexity

The above model was implemented and solved with the same rule set used in Section 5.4. A Java program was written to read the rules from a file in the MAM evaluation function format and generate the SAT model. The SAT model was then solved using the open-source SAT solver *miniSAT* [ES03]. The problem had 23 literals and around 45 clauses[4], each instance of the problem required less than a thousandth of a second to solve. All instances were solved with propagation alone, no search was required.

This model was then used in conjunction with a Java program, which reads a rule set directly from a MAM system log file, generates the models and checks the rule set for redundancy. Using the actual hardware that MAM runs on, reading in and checking a rule set required approximately 5 seconds. The majority of this time was taken to read and parse the log file. Each individual SAT model required approximately 15 thousandths of a second to solve. It is clear that the specialised SAT model offers improvement of $5-6$ orders of magnitude for redundancy checking over the general Promela model.

[4] The number of clauses is dependant on the rule being checked.

7 Overlapping rules

We have defined a rule to be redundant if it can be removed from a system without affecting how the system operates. However, a rule may also be redundant if it serves no *useful* purpose. For example, a rule set may include the two rules "If I receive a message from Bill then play a bird noise earcon" and "If I receive a message from Bill then play a doorbell earcon". At a system level these rules are different. However, they both play a sound when a message is received from Bill.

One could interpret this as a lack of confluence, i.e. we have overlapping left hand sides of rules and divergent right hand sides. Rules can also overlap in more subtle ways (e.g. a form of superposition). For example, a rule set may include the two rules "If the red or yellow buttons are pressed play the doorbell earcon and send a text message to Bill" and "If the red or blue buttons are pressed play the doorbell earcon and inform me using synthesised speech". Both rules cause the doorbell earcon to be played, when one condition is satisfied.

While these are only simple examples, they raise the question of what exactly we should be looking for when detecting redundant rules and what is the underlying theory of modalities? Moreover, to answer these question we need to know why we are interested in this problem. For example, is it a significant issue in practice?

The answer to the latter appears to be positive. While conducting user evaluation studies, the developers of the MAM system have found that many test subjects indicated they have trouble understanding complex rules and only want to define simple rules. This means there is significant scope for overlapping conditions and actions. Therefore, it would be advantageous for the system to offer assistance. This could be in the form of a message informing the user that a rule they entered is redundant, or makes some other rule redundant, or is overlapping with another rule. Alternately, the system may simply detect such rules and only partially implement them. In any case, further study is need to understand user intentions and their relation to modalities and context, and also how best to feedback information from any analysis.

8 Discussion

A distinctive feature of this work is we are trying to more tightly couple design with modelling, closing the loop between design, use, configuration, modelling and verification. Further, we deal with systems as actually deployed, rather than an ideal yet to be implemented.

A long term goal is to automate many of these processes and so in this case study, where possible, we have developed scripts to process inputs automatically e.g. log files.

The general model is based around the central concept of event – the MAM system is after all event-driven. It captures a wide range of functional, temporal behaviour. However, in the context of checking for redundancy within rules, complexity of the model became a concern, especially if we aim for real-time model-checking in a live MAM. Furthermore, it is not clear that given the form of rules in this application, analysis of the rules requires a temporal logic. To a great extent, in this application, one could argue that the state of a sensor encapsulates a set of computational paths (or at least what is required to know them) and so we do not need to study the paths themselves. So, a SAT model is appropriate for this type of verification. Furthermore,

the verification then became so efficient it could be applied directly within the MAM, running in real time on the same computational hardware.

While we have only investigated one of the properties we mentioned Section 5, redundancy, how we detect and resolve redundancy depends also on our understanding of modalities, priorities, and more generally, context. For example, a user may not care about overlapping earcons *unless* one of them has been generated by a certain condition. For example, delivering messages via the television and the beeper simultaneously may be acceptable, unless one of the messages is considered significantly more urgent than the other, or has arisen because of an unsafe context. The area of semantics and ontologies for modalities/context requires further investigation. Acceptability and usability of modalities may be regarded as an example of crossing the "semantic rubicon"[5] [KA02]. A contribution of our formal modelling and analysis to MAM design has been to expose this crossing.

9 Related Work

Much formal analysis of pervasive systems is focussed on techniques for requirements involving location and resources, within a waterfall framework. For example, [CD09] employs the Ambient calculus for requirements and [CE07] employs a constraint-based modelling style and temporal logic properties. Some work has been done on better integration of formal analysis techniques within the context of interactive system interfaces (e.g. [CH08]), but there is little work on more tightly coupled models and analysis. One exception is [RBCB08], where a model of salience and cognitive load is developed and a usability property is considered. The model is expressed in a higher order logic, and the property is expressed in LTL. While our paradigm is different, the authors recognise they are engaged in a cyclic process. In some cases, their verification revealed inconsistencies between experimental behaviour and the formal model, which led them to suggest refinements to the rules and also new studies of behaviour. Finally, there is ongoing work to use policy conflict handling mechanisms embedded in telecommunications systems in homecare applications [WT08]. We believe our approach can provide a more generic framework for such a conflict management service.

10 Conclusions and future work

We have considered the problem of verifying context-aware, pervasive, interactive systems. These kinds of systems present numerous challenges for verification: context-changes may be difficult to model and predict and in addition, such systems often operate in situations that require practically unpredictable changes to the application functionality itself. We have outlined an approach to verification that makes explicit two different types of interaction: system configuration and system use. Our long term goal is to more tightly couple reasoning about configurable systems by configuring models, and closing the loop between design, use, configuration, modelling and verification. In particular, we are concerned with feeding back results of verification to users, designers, configurers, and modellers.

[5] the division between system and user for high level decision-making or physical-world semantics processing.

This paper reports on preliminary results from an example concerning an activity monitor from the MATCH homecare system. We have developed an event based general model in Promela, formulated and checked a number of properties in the model checker SPIN. We have concentrated on supporting end user configuration by checking for rule redundancy. Results from the general model led us to develop a specialised model for use with a SAT solver, and using that model we were able to verify an example set (taken from an actual log file), on the actual MAM, in real time.

The case study illustrates a number of engineering and foundational challenges for our approach: we have not modelled an idealised system, but one that has been designed and engineered in the context of specific practices and personal conventions. This presents non-trivial challenges for any modelling process. The work is still preliminary, but our results demonstrate proof of concept. A distinctive feature of the work is we generate automatically parts of the model from actual log files.

Longer term, our plans for further future work include generating more parts of models automatically from log files, for a class of context aware systems, and incorporating aspects of stochastic behaviour, performance, and real-time in the model and properties. We also plan to further investigate semantic models of modalities and context and the best way to present and use verification results, expecially in the context of human and non-human agents.

Acknowledgements:

This research is supported by the *VPS* project (*Verifying interoperability in pervasive systems)*, funded by the Engineering and Science Research Council (EPSRC) under grant number EP/F033206/1. We also acknowledge support from the MATCH Project, funded by the Scottish Funding Council under grant HR04016.

We also acknowledge the work of Tony McBryan, the designer and implementer of the MAM System, who kindly provided the log files we used and offered technical assistance.

Bibliography

[CD09] A. Coronato, G. De Pietro. Formal specification of a safety critical pervasive application for a nuclear medicine department. *International Conference on Advanced Information Networking and Applications Workshops*, pp. 1043–1048, 2009. doi:10.1109.WAINA.2009.198

[CE07] A. Cerone, N. Elbegbayan. Model-checking Driven Design of Interactive Systems. *Electron. Notes Theor. Comput. Sci.* 183:3–20, 2007. doi:dx.doi.org/10.1016/j.entcs.2007.01.058

[CH97] J. Campos, M. D. Harrison. Formally verifying interactive systems: a review. In *Design, Specification and Verification of Interactive Systems 97*. Pp. 109–124. Springer, 1997.

[CH08] J. C. Campos, M. D. Harrison. Systematic analysis of control panel interfaces using formal tools. In *XVth International Workshop on the Design, Verification and*

Specification of Interactive Systems (DSV-IS 2008). Lecture Notes in Computer Science 5136, pp. 72–85. Springer-Verlag, July 2008.

[CRB07] P. Curzon, R. Rŭkėnas, A. Blandford. An approach to formal verification of human-computer interaction. *Formal Aspects of Computing*, pp. 513–550, 2007. doi:10.10.1007/s00165-007-0035-6

[ES03] N. Eén, N. Sörensson. An Extensible SAT-solver. In Giunchiglia and Tacchella (eds.), *SAT*. Volume 2919, pp. 502–518. Springer, 2003.

[Hol03] G. J. Holzmann. *The SPIN model checker: primer and reference manual*. Addison Wesley, Boston, 2003.

[HS99] G. Holzmann, M. H. Smith. Software model checking - Extracting verification models from source code. In *Proc. FORTE/PSTV '99*. Pp. 481–497. Kluwer, 1999.

[JAK] Jake Project. http://code.google.com/p/jake-drivers/

[KA02] T. Kindberg, F. A. System software for ubiquituous computing. *Pervasive computing*, pp. 70–81, 2002.

[MG09] T. McBryan, P. Gray. User Configuration of Activity Awareness. *Lecture Notes In Computer Science* 5518:748–751, 2009. doi:dx.doi.org/10.1007/978-3-642-02481-8_113

[MRM09] P. Markopoulos, B. de Ruyter, W. E. Mackay. *Awareness Systems: Advances in Theory, Methology and Design*. Springer, 2009. doi:10.1007/978-1-84882-477-5

[RBCB08] R. Rŭkėnas, J. Back, P. Curzon, A. Blandford. Formal modelling of salience and cognitive load. *ENTCS*, pp. 57–75, 2008. doi:10.10.1016/j.entcs.2008.03.107

[RM07] Y. Riche, W. Mackay. MarkerClock: A Communicating Augmented Clock for the Elderly. *Proc. Interact 07. Part II, Lecture Notes In Computer Science* 4663:408–411, 2007.

[SHA] Shake users group. http://www.dcs.gla.ac.uk/research/shake/

[WMH07] J. Williamson, R. Murray-Smith, S. Hughes. Shoogle: excitatory multimodal interaction on mobile devices. *Proc. SIGCHI Conference on Human Factors in Computing Systems* 4663:121–124, 2007. doi:http://doi.acm.org/10.1145/1240624.1240642

[WT08] F. Wang, K. Turner. Policy Conflicts in Home Care Systems. *Proc. 9th Int. Conf. on Feature Interactions in Software and Communications Systems*, pp. 54–65, 2008.